# Planning your Project

## ECM28 computing school

## 23rd August 2013

Airlie McCoy

University of Cambridge, CIMR

UNIVERSITY OF CAMBRIDGE

Parts of this talk are inspired by Tim Love's talk at www-h.eng.cam.ac.uk/help/tpl/talks/projman.php

# What is a software project?

Some or all of:

- Countless lines of code
- Years of development
- More than one developer
- Thousands of users
- Years of use
- Hundreds of citations
- Supported explicitly by grant(s)
- Revenue stream from licensing

*Antonym: jiffy*

# Project Management

- Microsoft survival guide:

  "Of the most expensive software projects, about half will eventually be cancelled for being out of control. Many more are cancelled in subtle ways."

- Oxford University/Computer Weekly:

  "Only 16% of IT projects were considered successful"

- The Standish CHAOS report:

  "the software success rate is 24% overall, with numbers even lower for large projects, especially those in the government sector"

# Software Engineering

Cambridge University
Computer Science Tripos
Part IB

At the end of the course

- You should know how writing programs with tough assurance targets and/or in large teams differs from small programming exercises
- You should appreciate the waterfall, spiral and evolutionary models of development and be able to explain which kinds of software development might profitably use them
- You should appreciate the value of tools and the difference between incidental and intrinsic complexity
- You should understand the basic economics of the software development lifecycle

# Academic Software Engineering

You are both customer and vendor

Most problems in successful delivery of software arise from customer nescience & misunderstandings between customer and vendor

# Academic Software Engineering

- Software may be written as a one-off yet end up being used for years
- Software is unlikely to depend on commercial libraries
- Software may only ever be used in house
- Software may well result in freeware
- Software may well be written by one person
- Software may have cutting edge technology
- Software may not have much of a GUI

How can we adapt Software Engineering procedures to this situation?

# Overview

Managing Software Projects

- Language and Libraries
- Productivity
- Bugfixes
- Release
- Legacy

# Languages

- Scripting languages are good for development speed
  - Slow execution speed
- Compiled languages are good for execution speed
  - Arguably, slower to develop
- Use what others are using
  - You may not have a choice – lab policy
  - You may not have a choice – library use
    - e.g. cctbx is in c++ with python wrappers

# Libraries

## THE GOOD

- Use the standard libraries in the languages
  - e.g. stl standard template library in C++
- Use libraries so that you don't reinvent the wheel
  - cctbx symmetry libraries
  - mtz reading libraries
- Use libraries that everyone else is using
  - This acts as a check on their bugginess

## THE BAD

- Be hyper-aware of the licensing conditions of the libraries you use
  - Don't use commercial libraries
- Avoid libraries that contain massively more that you need
  - e.g. the whole NAG library for a single matrix inversion routine
- Avoid libraries that are bound to an operating system

# Plug-in architectures

- Splitting the task into de-coupled sections has many benefits
  - Others can extend the program
  - Developers/teams can work independently
  - Decisions about adding features can be deferred
- This is the aim of object oriented languages
  - *But have they succeeded?*
  - *Objects have failed: Notes for debate*

    *by Richard P. Gabriel*

    "We find that object-oriented languages have succumb (sic) to static thinkers who worship perfect planning over runtime adaptability, early decisions over late ones, and the wisdom of compilers over the cleverness of failure detection and repair."


**"The determined Real Programmer can write FORTRAN programs in *any* language"**

*Ed Post*

*Real Programmers Don't Use Pascal (1982)*

# Productivity

# The fun bit

# … is writing new code

What about the rest?

# Programmers

- "Good programmers are 10 times more productive than bad"
  - "no conventional salary structure provides for this kind of dynamic range" (Microsoft survival guide)
- PIs will often give a good/busy programmer a new task rather than temporarily deploy/employ a student/postdoc
- If a programmer is seconded, there may be failures elsewhere as a consequence
  - How much task-balancing is your responsibility?
- Adding staff can slow down a project

# IDE

- You can use an IDE
  - Integrated Development Environment
- "Unix is the perfect IDE"
  - vi, emacs, sed
    - Fast search/replace, file opening/closing
    - "Think-speed" editing
  - find, grep, locate
  - Shell scripts for scripting editing etc
  - No window bloat, menu hunting
    - No pointing and clicking with mouse


# The editor wars

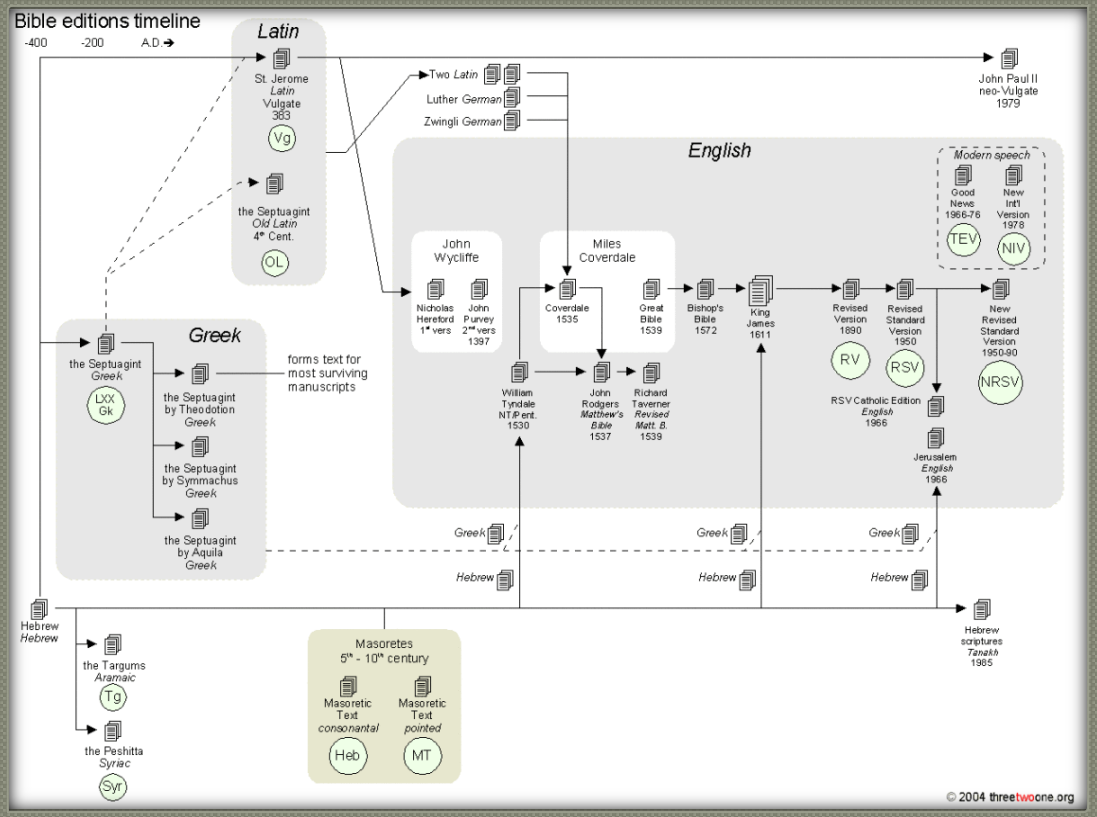
# Should I use Vi?

"What language are you writing…? If you're writing Delphi then I don't think VIM is going to help you. If you're writing a C# app, maybe you're better off in Visual Studio. Java? Maybe stay in Eclipse. But for many languages, like C, Perl, Python, etc, people report they are more productive in a text-only editor (like VIM or EMACS)."

Warren P.
August 27th 2912 20.52
Stackexchange

# Revision Control

- Also called Version or Source Control
- At its most basic, each recorded change to the code is given the next revision number in sequence,
  - starting at revision 1
- Handles software management
  - Timestamps
  - Reversion to earlier states of the code
  - Comparisons of different revisions
- Revision control boosts your productivity




*The Septaugint Greek*
*The Septaugint by Old Latin*
*The Septaugint by Aquila*
*Luther German Bible*
*Saint Gerome Latin Vulgate*
*Tyndale Bible*
*Great Bible*
*Bishop's Bible*
*King James Bible*
*Revised Version*
*Revised Standard Version*
*New Revised Standard Version*
*Good News Bible*
*Revised International Version*

**The need for revision control has existed since the invention of writing**

"Editions"

Revision control is embedded in may software applications
e.g. Word (undo, track changes)
e.g. Wikipedia

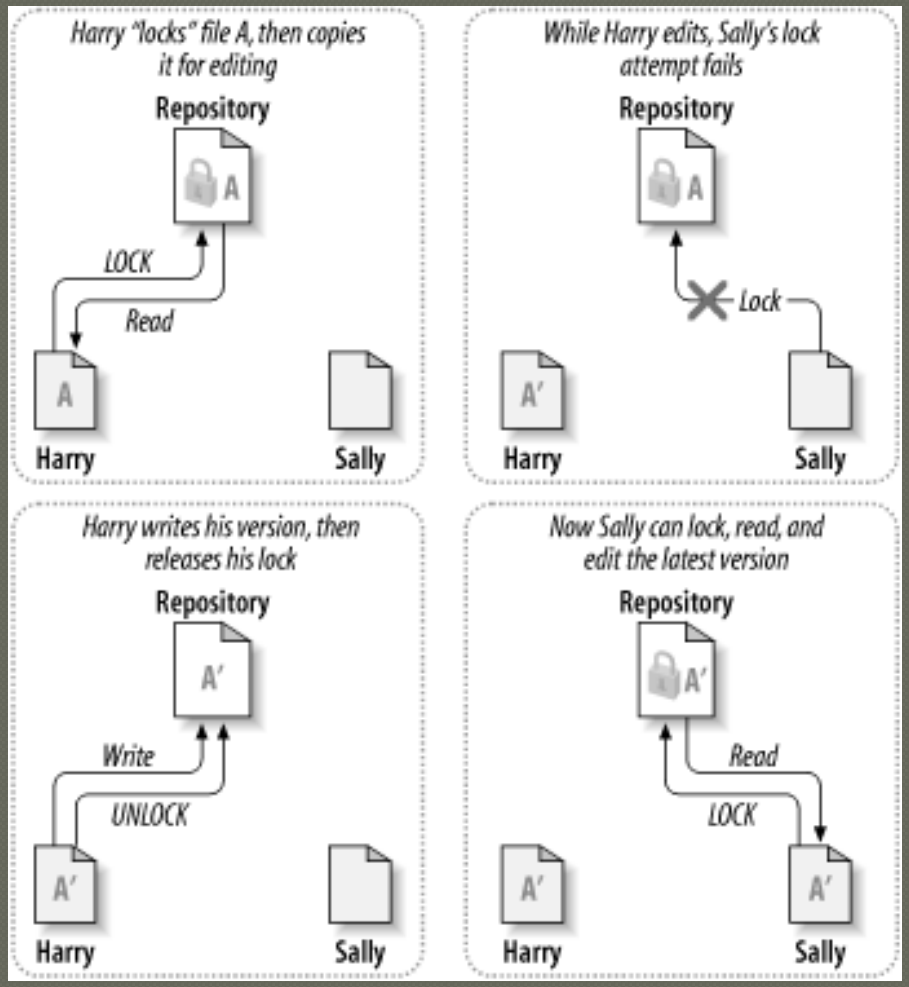
**Revision control became more fine-grained with the development of computing**

It also became an essential tool for software management
and a major field in its own right

# Which system?

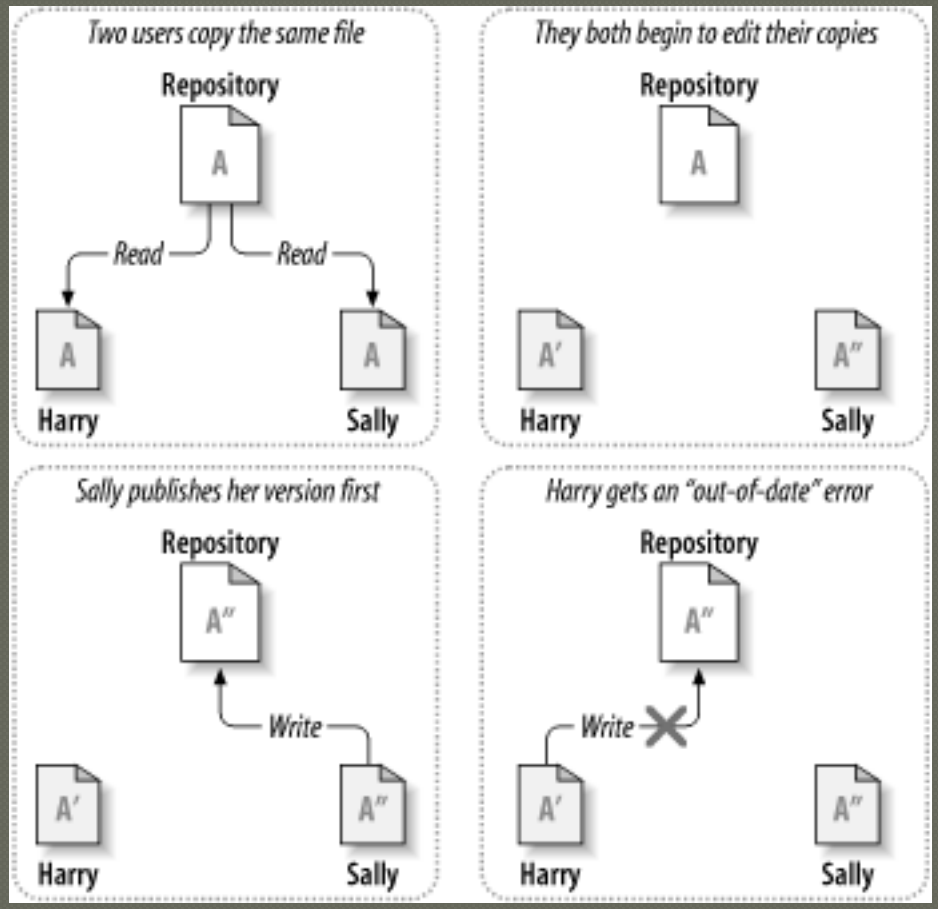
- Copy-Lock-Modify:
  - Microsoft Visual SourceSafe (VSS)
- Copy-Modify-Merge:
  - Central Repository
    - CVS
    - subversion (aka svn)
  - Distributed
    - git
    - bazaar

# Lock-Copy-Unlock




- Inhibits parallel development
- Conflicts easily generated
  - Harry modifies A, Sally modifies B but together the modifications cause a bug
  - No responsibility for clashes
- Good for binary files

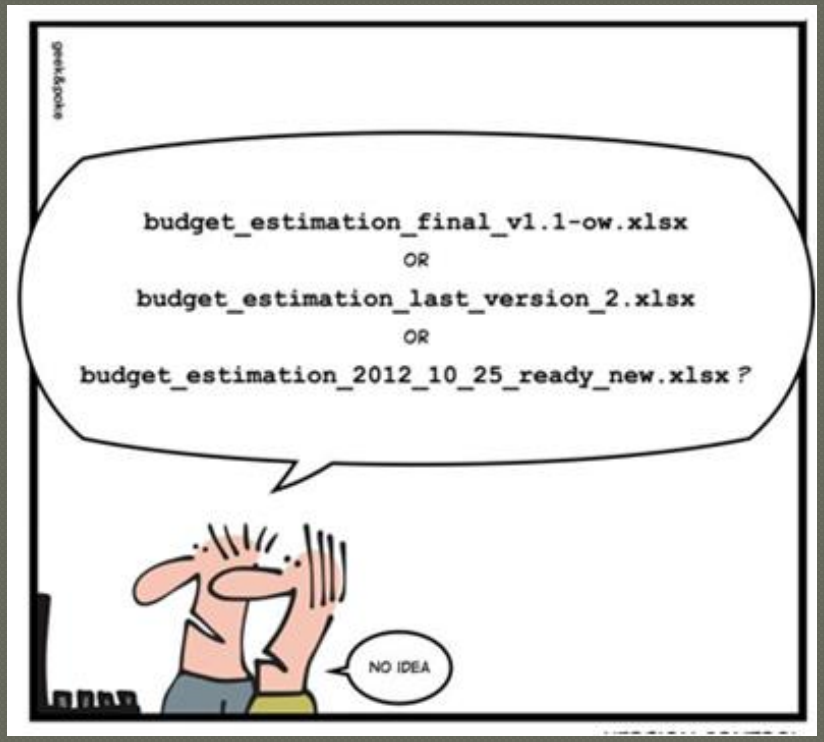
# Copy-Modify-Merge




- Facilitates parallel development
- Conflicts detected automatically
  - Harry modifies A, Sally modifies B but "last one in" has responsibility to fix clashes before adding their changes
- Poor for binary files

# Working with SVN

- trunk
  - Current releasable version
  - Must compile
  - "Don't work in the trunk"
- tags
  - Releases and versions
  - create once
  - never modified
- branches
  - each branch can be a disjoint component of your architecture
  - works best with loosely coupled, highly OO design
  - requires code revision and integration testing



# SVN for single developer

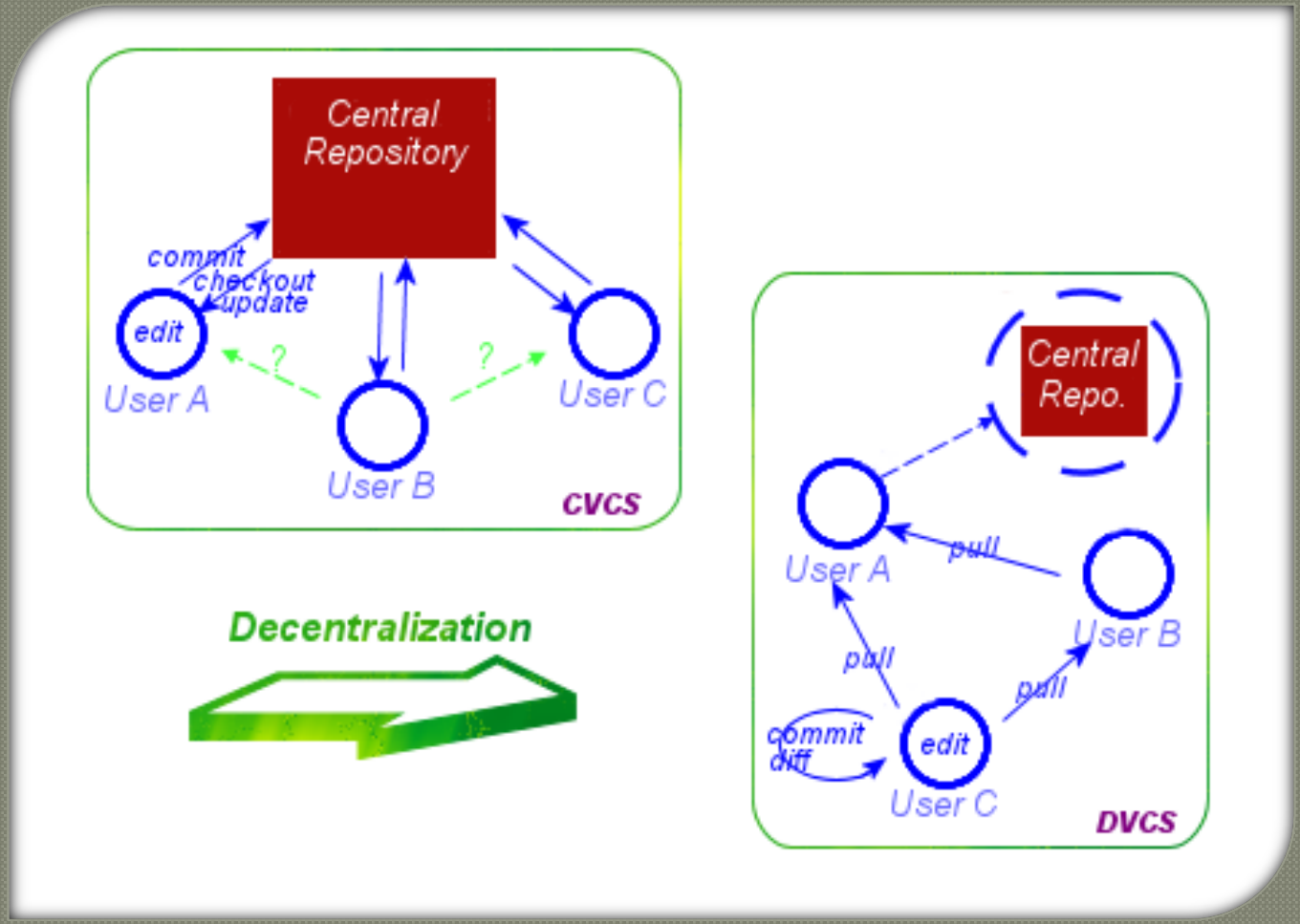
- Get to work
- Sit at your desktop computer and login
- Read email
- Find bug report from a guy in the US
- Work on bugfix all day without solving problem but having found some typos and added a small feature from the todo list
- Realise the problem while cycling home
- Finish bugfix at home
- Save the changes
- Make your bugfix available to the guy in the US
- Let everyone in group know a bugfix is available
- Go to sleep

# SVN for single developer

- You can work in the trunk
  - don't branch, although tags may be useful
- Use as a complete digital notebook
- Facilitates code development on multiple platforms/computers
- Code changes are easily available to users who may have requested a bugfix/feature
- Can set up automatic email notification of changes to "interested parties" e.g. CCP4
- If the repository is off site, you get off site backup automatically

# Drawbacks of SVN

- No history-aware merge capability, making merging of branches error prone
  - Fails to merge changes if files are renamed
- No way of pushing changes to another developer without committing to the central repository
- Offline commits are not possible
  - No "personal" repository
  - Code must compile and have passed regression tests to be committed




# Decentralized Version Control

Major advantages for projects with many "equivalent" developers

# Increasing your Productivity

- Don't go to conferences
- Don't go to developers' meetings
- Don't write papers
- Don't have coffee breaks…

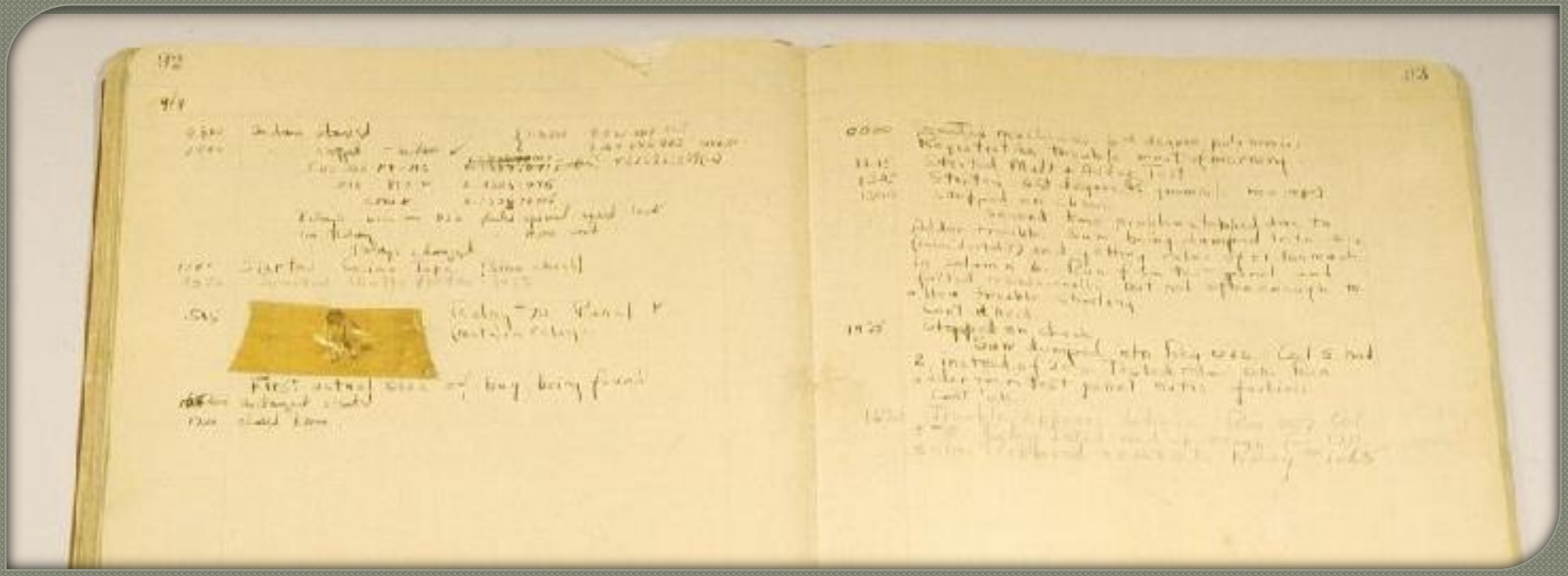
# Productivity and Working Alone

- For much of the time you're likely to be working alone and unsupervised
- You may have no-one to bounce ideas off
- You may have no-one who is interested in your progress
- You need to manage your strengths and weaknesses

# Productivity and Working Alone

- Keep different kinds of work available so that there's always something to match your mood
  - Bugfixes, new features, documentation, literature
- Be aware of your 'displacement activities'
  - Try to make them something useful (documentation).
  - A common diversion is to over-develop tools and utilities that are not important/useful/part of the project
- Make a list of easy things that you can tick off
- Beware of Burnout
  - Sometimes “less is more”
- Communicate with developers in other labs
- Keep the goal of the project in mind

# Bugfixes



"Things were going badly; there was something wrong in one of the circuits of the long glass-enclosed computer," she said. "Finally, someone located the trouble spot and, using ordinary tweezers, removed the problem, a two-inch moth. From then on, when anything went wrong with a computer, we said it had bugs in it."

## The first computer bug

U.S. Navy Capt. Grace Murray Hopper
Logbook 1947
National Museum of American History

# Bugs

- Microsoft Survival Guide:

  “industry average experience suggests that there are 15-50 errors per 1000 lines of delivered code”

- Computer Weekly:

  "Despite testing procedures, even a best-of-breed product usually still contains a margin of error, typically around 5%. At this point it may be deemed by both vendor and customer that trying to reduce this percentage is a case of diminishing returns and not worth the additional Investment."

# How to find the Bugs

- Don't introduce bugs in the first place
  - Know the project and the code backwards
- Code compilation
  - Huge advantage of compiled over scripting languages
- Unit and Regression tests
  - Makes sure that all features *in the test suite* work correctly
  - Stop old bugs being reintroduced
- Bug reports from users
  - Litter the code with assert statements where there could be numerical exceptions, and print your email address if they are triggered
- Power users may send bug report and bugfix together

# Off-the-wall Bug Ideas

- Code walkthrough
  - Get someone to listen as you describe the code
- Use "defect seeding"
  - Put 10 deliberate errors into your code or documentation and get someone to check it. If they find 20 errors but only 5 of them are your deliberate ones then you might hypothesise that since they only found half of the deliberate errors they only found half of the other errors, so there are 30 unintentional errors
  - Some people find this a fun way to work
  - Don't forget to remove seeded bugs!
- Run a bug-finding competition
  - e.g. LATEX documentation prize for most number of bugs

# Testing

- Testing is the process of validating and verifying that the software
  - works as expected – bugs/features
  - meets the requirements that guided its design and development – publications/grants
- Testing can never completely identify all the defects
- Testing cannot establish that software functions properly under all conditions but can only establish that it does not function properly under specific conditions

Testing is a trade-off between budget, time & quality

## Project Management Triangle


- Design something quickly and to a high standard, but it will not be cheap.
- Design something quickly and cheaply, but it will not be of high quality.
- Design something with high quality and cheaply, but it will take a long time.

# Test Suite

- You must have a test suite
  - Basic component of software development
- Test suites can be very boring to set up
  - Set up data
  - Set up running jobs
  - Set up result checking
  - Set up database of results/display
- Range from minutes to days of CPU to run
- Good, large test suites can and very often do reveal interesting science

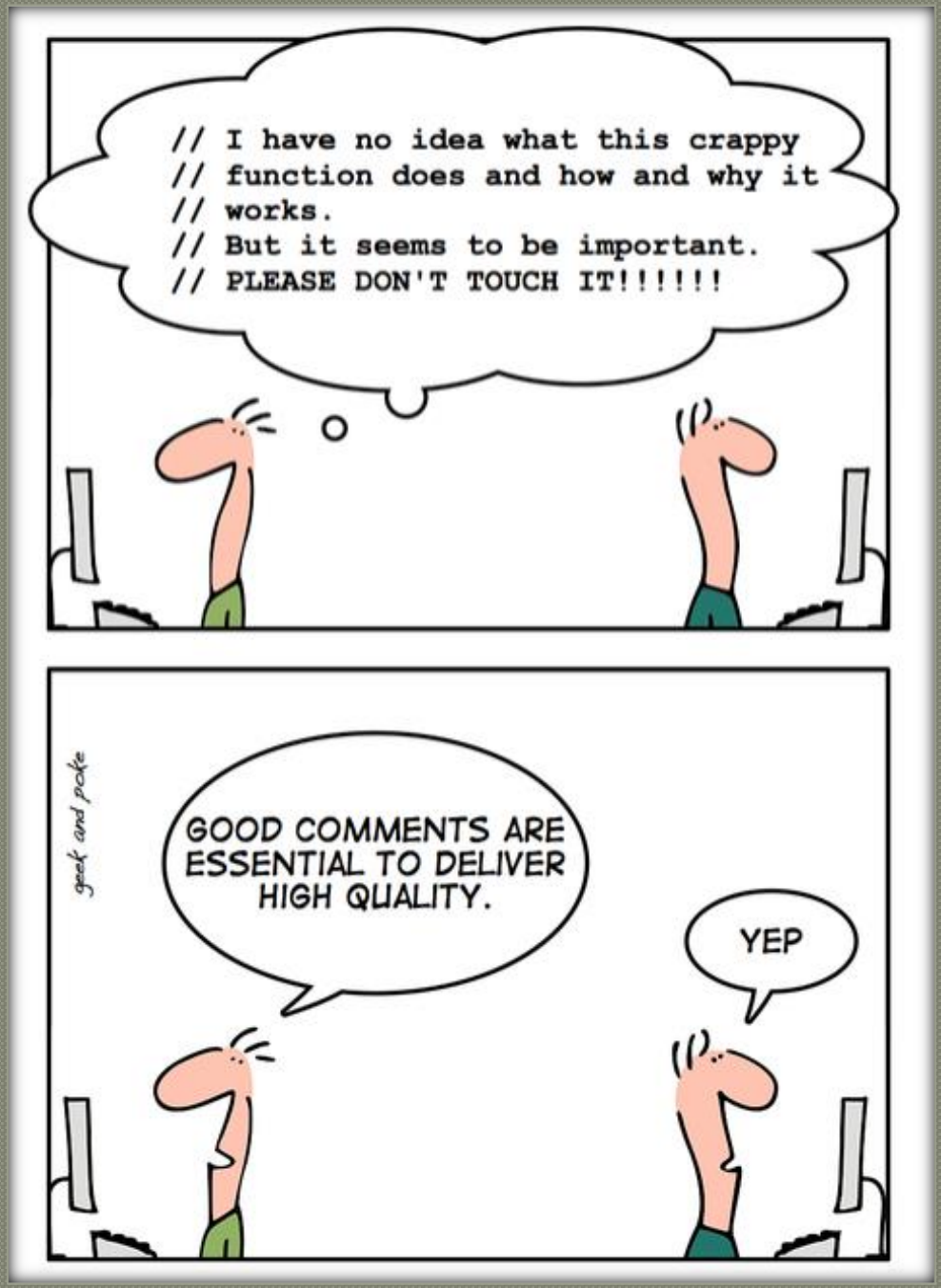
# Bugs and Web applications

- Web applications may present difficulties because of the number of components that may be involved
- If there's a problem (in particular performance-related) the cause could be:
  - the network
  - the database
  - the web server (with modules for PHP support)
  - the web proxy-server or
  - the browser itself (which comes in various varieties each with different plug-ins, java interpreters …).

# Release

# Documentation

- Misunderstandings can arise as soon as the program needs to be used by someone other than the programmer
- Documentation, more documentation, and even more documentation
  - Self-documenting code
  - Comments in code
  - Website
  - Publications
  - Tutorials
  - Seminars




# Documentation

# Time estimates

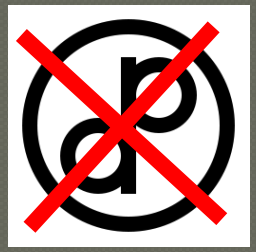
- Hard
  - One of the aims of software engineering is to improve the accuracy of these estimates
- Decomposing the task helps
  - If tasks are small then they're more likely to be comparable, making future estimation easier
- 80% of time is spent on unplanned rework
- The industry average for code production is 8-20 lines of correct code per day
  - independent of the language
- New programmers tend to be over-optimistic
  - More experienced ones tend to "sandbag"

# Release Schedule

- May not be your decision
  - e.g. CCP4, phenix release schedule
- Preceded by a "code freeze"
  - No new features
  - Only bugfixes allowed
  - Can go on for far longer than intended

# Licensing

- Before release you will have to license the software
  - You cannot put the code in the public domain
    - Public domain makes works a free-for-all e.g. the works of Shakespeare
  - Public domain is only for works for which the copyright has EXPIRED
  - Copyright holders cannot give up copyright
  - Therefore, any user but yourself will need to know under what conditions they can use the software



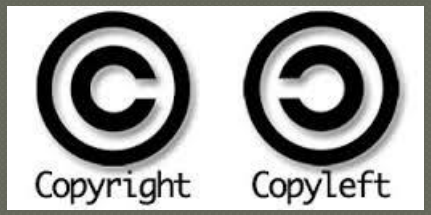
# Copyright

- Licensing is the prerogative of the copyright holder
- The creator of the work owns the copyright
  - No need to register copyright, automatic
  - Lasts until 70 years after death of creator
- EXCEPT if the work is created in the course of employment
  - Your employer owns the copyright
  - Even if the work is done in your "spare time"
  - Lasts for 120 years from the date of creation

# Free Software

GRATIS


LIBRE




# License Types

- Copyleft
  - Offers the right to distribute copies and modified versions of a work and requiring that the same rights be preserved in modified versions of the work
  - Copyleft licences are VIRAL LICENCES
- Weak copyleft
  - Used for the creation of software libraries
  - Only changes to the weak-copylefted library itself become subject to the copyleft, not the linked software
- Non-copyleft
  - Do not require the licensee to distribute derivative works under the same license

# Libre

- There is an ongoing debate as to which class of license provides the greater degree of freedom
- This debate hinges on issues such as the definition of freedom and whose freedoms are more important
- Copyleft maximizes the freedom of all potential future recipients of a work (freedom from the creation of proprietary software)
- Non-copyleft free software licenses maximize the freedom of the initial recipient (freedom to create proprietary software)

# Legacy

# Distribution

- Distribution is where previous decisions may come back to haunt you
  - Will it be trivial to port?
  - Are the libraries suitable for distribution?
- Source code/executables
  - If source code you will need to distribute build system also
  - If executable, must supply it for (specified) OS
- There are MANY routes to distribution
  - "email me"
  - CCP4
  - Program specific website
  - Start a company to handle distribution

# Free Software

GRATIS


LIBRE

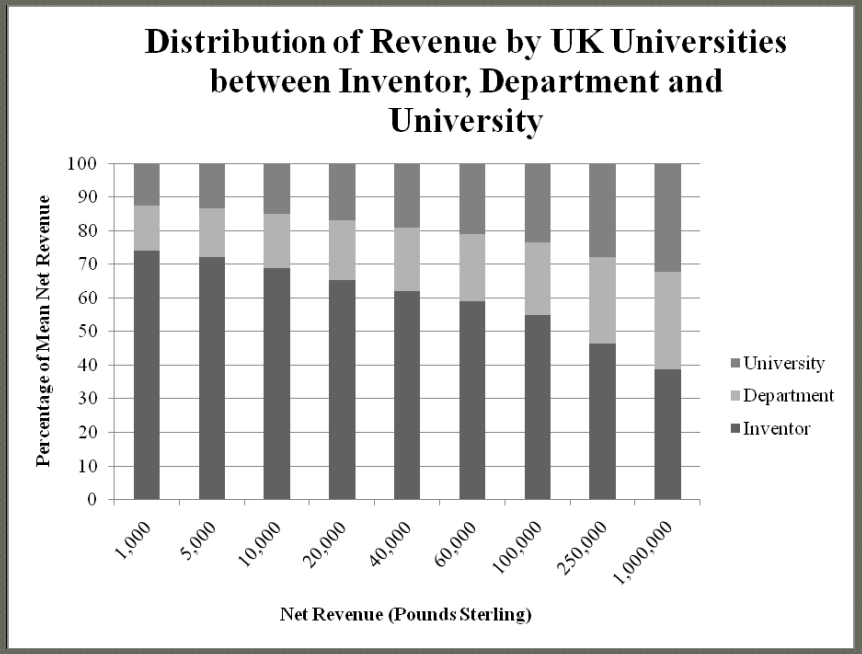
# Gratis

- Your institution will have a standard revenue sharing policy
  - The employer will get most of the money
  - True even if you leave your employer
- Grant terms may involve revenue sharing with the grant agency

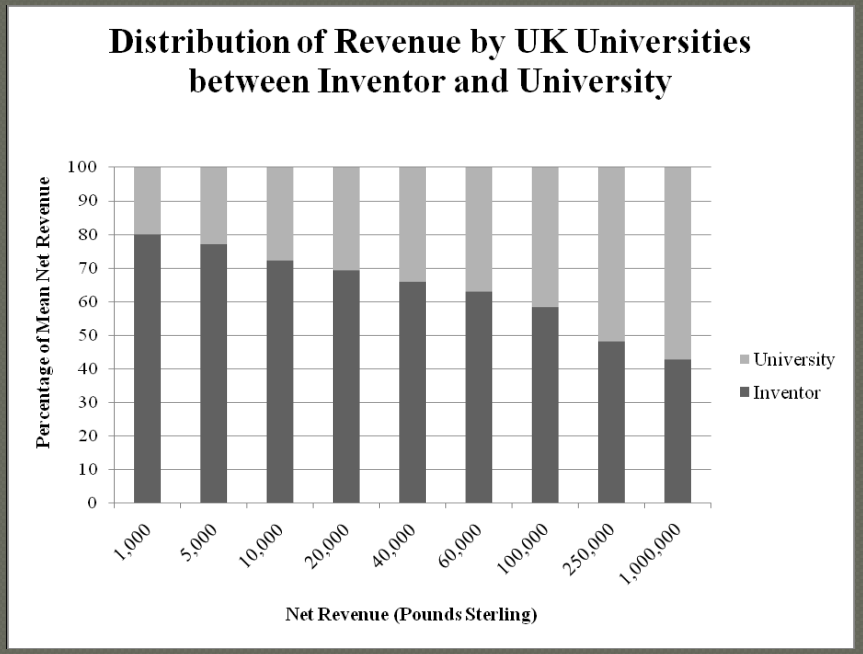
| Net Revenue | Inventor(s) | Department(s) | University |
|---|---|---|---|
| **Up to £10,000** | 75% | 12.5% | 12.5% |
| **£10,000 - £60,000** | 65% | 17.5% | 17.5% |
| **£60,000 - £250,000** | 50% | 25% | 25% |
| **Over £250,000** | 40% | 30% | 30% |

# Revenue Sharing Averages: UK

## THREE-WAY SHARING




## TWO-WAY SHARING




## Licensing

There will be a lawyer involved at some point

# Legacy

- Maintenance is
  - "65% new requirements"
  - "18% changes for new OS"
  - "17% bug fixes"
- Your software may end up being wrapped in another application
  - Issues of accreditation arise
  - Others will want to "fix" the version your software at the one they used for development and are not interested in your later work
- *Your software will be superseded*